AF/IFW

GUNNISON, McKAY & HODGSON, L.L.P.
GARDEN WEST OFFICE PLAZA, SUITE 220
1900 GARDEN ROAD
MONTEREY, CALIFORNIA 93940
(831) 655-0880
FACSIMILE (831) 655-0888

OIPE MAP2a
DEC 08 2008
PATENT & TRADEMARK OFFICE

December 3, 2008

Mail Stop Appeal Brief-Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

TRANSMITTAL LETTER FOR REVISED APPEAL BRIEF

RE: Applicant(s):	Judith E. Schwabe et al.		
Assignee:	Sun Microsystems, Inc.		
Title:	OPTIMIZATION OF N-BASE TYPED ARITHMETIC INSTRUCTIONS VIA REWORK		
Serial No.:	10/712,463	Filed:	November 12, 2003
Examiner:	Tuan A. Vu	Group Art Unit:	2193
Docket No.:	P-4181CIP		

Dear Sir:

Transmitted herewith are the following documents in Response to the Notice of Non-Compliant Appeal Brief dated November 5, 2008 in the above application:

1. Return receipt postcard;
2. This Transmittal Letter (2 pages); and
3. Revised Appeal Brief (59 pages)

☒ Conditional Petition for Extension of Time: If an extension of time is required for timely filing of the enclosed documents after all papers filed with this transmittal have been considered, Applicant(s) hereby petition for such an extension of time.

Transmittal Letter
Serial No. 10/712,463
December 3, 2008

☒ The Commissioner is hereby authorized to charge any additional fees required for consideration of the enclosed documents, and to credit any overpayment of fees to Deposit Account No. 50-0553.

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on December 3, 2008.

Attorney for Applicant(s) — December 3, 2008 Date of Signature

Respectfully submitted,

Forrest Gunnison
Attorney for Applicant(s)
Reg. No. 32,899

OIPE IAP22
DEC 08 2008
PATENT & TRADEMARK OFFICE

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s):	Judith E. Schwabe, Zhiqun Chen		
Assignee:	Sun Microsystems, Inc.		
Title:	OPTIMIZATION OF N-BASE TYPED ARITHMETIC INSTRUCTIONS VIA REWORK		
Serial No.:	10/712,463	Filed:	November 12, 2003
Examiner:	Tuan A. Vu	Group Art Unit:	2193
Docket No.:	P-4181CIP		

Monterey, CA
December 3, 2008

Mail Stop Appeal Brief-Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

REVISED APPEAL BRIEF

Dear Sir:

Appellant files this Revised Appeal Brief in response to the Notice of Non-Compliant Appeal Brief dated November 5, 2008.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

REAL PARTY IN INTEREST

The assignee of the above-referenced patent application, Sun Microsystems, Inc., is the real party in interest.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

RELATED APPEALS AND INTERFERENCES

No other appeals or interferences are known to the undersigned Attorney for Appellant, or the Assignee Appellant, which will directly affect, or be directly affected by, or have a bearing on the Board's decision in this pending Appeal.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

STATUS OF CLAIMS

Claims 1 to 78 are pending. Claims 1 to 78 stand rejected in the Final Office Action of March 27, 2008. The rejections of Claims 1 to 78 are hereby appealed.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

STATUS OF AMENDMENTS

All amendments to the claims presented by Appellant have been entered.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

SUMMARY OF CLAIMED SUBJECT MATTER

A summary is provided below for each independent claim and for each dependent claim argued separately.

CLAIM 1

With respect to Claim 1, a method for arithmetic expression optimization includes validating, converting and matching. In the validating, at least one input stack associated with a first instruction configured to operate on at least one operand of a first type is validated {1615, Fig. 16; Lines 11 to 13, Paragraph [0062]; See also Fig. 17 and Paragraph [0063] that further describe operation 1615; Paragraph [0064] and Fig. 18 that further describe operation 1700 of Fig. 17.}. Each of the at least one input stack is associated with an input instruction of the first instruction. {Paragraph [0059]; lines 4 to 7, Paragraph [0062].} Each input stack represents the state of an operand stack associated with an input instruction upon execution of the input instruction. {See Paragraphs [083], [089], [0131], [0135], [0141] for examples of operation 1615. See also Paragraphs [0011], [0012] for a background discussion of operand stacks}.

In the converting, the first instruction is converted to a second instruction configured to operate on at least one operand of a second type {1625, Fig. 16; lines 13, 14, Paragraph [0062]; See also, Fig. 19 and Paragraph [0065] that further describe operation 1625.}. The second type is smaller than the first type and the converting is based at least in part on the relative size of the first type and the second type, wherein the second instruction is different from the first instruction {Paragraph [0065]; See also Paragraphs [083], [090], [0131], [0135], [0141] for examples of operation 1625.}.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

In the matching, the second type is matched with an operand type of at least one operand in the at least one input stack associated with the second instruction {1630, Fig. 16; lines 14 to 16, Paragraph [0062]; See also, Fig. 20 and Paragraph [0067] that further describe operation 1630; Fig. 21 and Paragraph [0068] that describe operation 2025 in Fig. 20; and Fig. 22 and Paragraph [0069] that further describe operation 2105 in Fig. 21}. The matching comprises changing the type of instructions in a chain of instructions to equal the second type if the operand type is less than the second type {Paragraph [0068]}. The chain is bounded by the second instruction and a third instruction that is the source of the at least one operand {Paragraph [0068]}. {See also paragraphs [084], [091], [106] to [0121], [132], [133] and [137] to [140] for examples of operation 1630.}.

CLAIM 11

With respect to Claim 11, Claim 11 includes the limitations of Claim 1 and further defines the converting of Claim 1. The converting further includes setting the second type to a smallest type {1900, Fig. 19; Paragraph [0065]}. The second type is set to the type of an operand in the at least one input stack if the smallest type is less than the type of the operand {1905, 1910, Fig. 19; Paragraph [0065]}. Also, the second type is set to a type that is larger than the smallest type {1935, Fig. 19; Paragraph [0065]} if the smallest type is greater than the type of the operand, {1915, Fig. 19; Paragraph [0065]} if the operand has potential overflow {1920, Fig. 19; Paragraph [0065]}, if the second instruction is sensitive to overflow {1925, Fig. 19; Paragraph [0065]} and if the second type is less than the first type {1930, Fig. 19; Paragraph [0065]}.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

CLAIM 20

With respect to Claim 20, a method for arithmetic expression optimization includes steps for validating, converting and matching. In the validating step, at least one input stack associated with a first instruction configured to operate on at least one operand of a first type is validated {1615, Fig. 16; Lines 11 to 13, Paragraph [0062]; See also Fig. 17 and Paragraph [0063] that further describe operation 1615; Paragraph [0064] and Fig. 18 that further describe operation 1700 of Fig. 17.}. Each of the at least one input stack is associated with an input instruction of the first instruction. {Paragraph [0059]; lines 4 to 7, Paragraph [0062].} Each input stack represents the state of an operand stack associated with an input instruction upon execution of the input instruction. {See Paragraphs [083], [089], [0131], [0135], [0141] for examples of operation 1615. See also Paragraphs [0011], [0012] for a background discussion of operand stacks}.

In the converting step, the first instruction is converted to a second instruction configured to operate on at least one operand of a second type {1625, Fig. 16; lines 13, 14, Paragraph [0062]; See also, Fig. 19 and Paragraph [0065] that further describe operation 1625.}. The second type is smaller than the first type and the converting is based at least in part on the relative size of the first type and the second type, wherein the second instruction is different from the first instruction {Paragraph [0065]; See also Paragraphs [083], [090], [0131], [0135], [0141] for examples of operation 1625.}.

In the matching step, the second type is matched with an operand type of at least one operand in the at least one input stack associated with the second instruction {1630, Fig. 16; lines 14 to 16, Paragraph [0062]; See also, Fig. 20 and Paragraph [0067] that further describe operation 1630; Fig. 21 and Paragraph [0068] that describe operation 2025 in Fig. 20;

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

and Fig. 22 and Paragraph [0069] that further describe operation 2105 in Fig. 21}. The matching comprises changing the type of instructions in a chain of instructions to equal the second type if the operand type is less than the second type {Paragraph [0068]}. The chain is bounded by the second instruction and a third instruction that is the source of the at least one operand {Paragraph [0068]}. {See also paragraphs [084], [091], [106] to [0121], [132], [133] and [137] to [140] for examples of operation 1630.}.

CLAIM 30

With respect to Claim 30, Claim 30 includes the limitations of Claim 20 and further defines the converting step of Claim 20. The converting step further includes a step for setting the second type to a smallest type {1900, Fig. 19; Paragraph [0065]}. The second type is set to the type of an operand in the at least one input stack if the smallest type is less than the type of the operand {1905, 1910, Fig. 19; Paragraph [0065]} in another step. Also, the second type is set to a type that is larger than the smallest type {1935, Fig. 19; Paragraph [0065]} if the smallest type is greater than the type of the operand, {1915, Fig. 19; Paragraph [0065]} if the operand has potential overflow {1920, Fig. 19; Paragraph [0065]}, if the second instruction is sensitive to overflow {1925, Fig. 19; Paragraph [0065]} and if the second type is less than the first type {1930, Fig. 19; Paragraph [0065]} in yet another step.

CLAIM 39

In Claim 39, a program storage device {Paragraph [0024]} readable by a machine, embodies a program of instructions executable by the machine to perform a method for arithmetic

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

expression optimization. The method for arithmetic expression optimization includes validating, converting and matching operations.

In the validating, at least one input stack associated with a first instruction configured to operate on at least one operand of a first type is validated {1615, Fig. 16; Lines 11 to 13, Paragraph [0062]; See also Fig. 17 and Paragraph [0063] that further describe operation 1615; Paragraph [0064] and Fig. 18 that further describe operation 1700 of Fig. 17.}. Each of the at least one input stack is associated with an input instruction of the first instruction. {Paragraph [0059]; lines 4 to 7, Paragraph [0062].} Each input stack represents the state of an operand stack associated with an input instruction upon execution of the input instruction. {See Paragraphs [083], [089], [0131], [0135], [0141] for examples of operation 1615. See also Paragraphs [0011], [0012] for a background discussion of operand stacks}.

In the converting, the first instruction is converted to a second instruction configured to operate on at least one operand of a second type {1625, Fig. 16; lines 13, 14, Paragraph [0062]; See also, Fig. 19 and Paragraph [0065] that further describe operation 1625.}. The second type is smaller than the first type and the converting is based at least in part on the relative size of the first type and the second type, wherein the second instruction is different from the first instruction {Paragraph [0065]; See also Paragraphs [083], [090], [0131], [0135], [0141] for examples of operation 1625.}.

In the matching, the second type is matched with an operand type of at least one operand in the at least one input stack associated with the second instruction {1630, Fig. 16; lines 14 to 16, Paragraph [0062]; See also, Fig. 20 and Paragraph [0067] that further describe operation 1630; Fig. 21 and Paragraph [0068] that describe operation 2025 in Fig. 20; and Fig. 22 and Paragraph [0069] that further describe

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

operation 2105 in Fig. 21}. The matching comprises changing the type of instructions in a chain of instructions to equal the second type if the operand type is less than the second type {Paragraph [0068]}. The chain is bounded by the second instruction and a third instruction that is the source of the at least one operand {Paragraph [0068]}. {See also paragraphs [084], [091], [106] to [0121], [132], [133] and [137] to [140] for examples of operation 1630.}.

CLAIM 49

With respect to Claim 49, Claim 49 includes the limitations of Claim 39 and further defines the converting of Claim 39. The converting further includes setting the second type to a smallest type {1900, Fig. 19; Paragraph [0065]}. The second type is set to the type of an operand in the at least one input stack if the smallest type is less than the type of the operand {1905, 1910, Fig. 19; Paragraph [0065]} Also, the second type is set to a type that is larger than the smallest type {1935, Fig. 19; Paragraph [0065]} if the smallest type is greater than the type of the operand, {1915, Fig. 19; Paragraph [0065]} if the operand has potential overflow {1920, Fig. 19; Paragraph [0065]}, if the second instruction is sensitive to overflow {1925, Fig. 19; Paragraph [0065]} and if the second type is less than the first type {1930, Fig. 19; Paragraph [0065]}.

CLAIM 58

With respect to Claim 58, an apparatus for arithmetic expression optimization [Paragraphs [0024] to [0026]; Fig. 5, Paragraphs [0032] to [0034]; and Fig. 6; paragraph [0035]} includes means for validating, means for converting and means matching.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

In the means for validating, at least one input stack associated with a first instruction configured to operate on at least one operand of a first type is validated {1615, Fig. 16; Lines 11 to 13, Paragraph [0062]; See also Fig. 17 and Paragraph [0063] that further describe operation 1615; Paragraph [0064] and Fig. 18 that further describe operation 1700 of Fig. 17.}. Each of the at least one input stack is associated with an input instruction of the first instruction. {Paragraph [0059]; lines 4 to 7, Paragraph [0062].} Each input stack represents the state of an operand stack associated with an input instruction upon execution of the input instruction. {See Paragraphs [083], [089], [0131], [0135], [0141] for examples of operation 1615. See also Paragraphs [0011], [0012] for a background discussion of operand stacks}.

In the means for converting, the first instruction is converted to a second instruction configured to operate on at least one operand of a second type {1625, Fig. 16; lines 13, 14, Paragraph [0062]; See also, Fig. 19 and Paragraph [0065] that further describe operation 1625.}. The second type is smaller than the first type and the converting is based at least in part on the relative size of the first type and the second type, wherein the second instruction is different from the first instruction {Paragraph [0065]; See also Paragraphs [083], [090], [0131], [0135], [0141] for examples of operation 1625.}.

In the means for matching, the second type is matched with an operand type of at least one operand in the at least one input stack associated with the second instruction {1630, Fig. 16; lines 14 to 16, Paragraph [0062]; See also, Fig. 20 and Paragraph [0067] that further describe operation 1630; Fig. 21 and Paragraph [0068] that describe operation 2025 in Fig. 20; and Fig. 22 and Paragraph [0069] that further describe operation 2105 in Fig. 21}. The matching comprises changing the type of instructions in a chain of instructions to equal

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

the second type if the operand type is less than the second type {Paragraph [0068]}. The chain is bounded by the second instruction and a third instruction that is the source of the at least one operand {Paragraph [0068]}. {See also paragraphs [084], [091], [106] to [0121], [132], [133] and [137] to [140] for examples of operation 1630.}.

CLAIM 68

With respect to Claim 68, Claim 68 includes the limitations of Claim 58 and further defines the means for converting of Claim 58. The means for converting further includes means for setting the second type to a smallest type {1900, Fig. 19; Paragraph [0065]}; means for setting the second type to the type of an operand in the at least one input stack if the smallest type is less than the type of the operand {1905, 1910, Fig. 19; Paragraph [0065]}; and means for setting the second type to a type that is larger than the smallest type {1935, Fig. 19; Paragraph [0065]} if the smallest type is greater than the type of the operand, {1915, Fig. 19; Paragraph [0065]} if the operand has potential overflow {1920, Fig. 19; Paragraph [0065]}, if the second instruction is sensitive to overflow {1925, Fig. 19; Paragraph [0065]} and if the second type is less than the first type {1930, Fig. 19; Paragraph [0065]}.

CLAIM 77

With respect to Claim 77, a method of using an application software program including arithmetic expression optimization of at least one instruction targeted to a processor is recited. The method includes receiving the software program on the processor (Fig. 5, Paragraphs [0031] [0032]}. The software

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

program is optimized according to a method including validating, converting and matching operations.

In the validating, at least one input stack associated with a first instruction configured to operate on at least one operand of a first type is validated {1615, Fig. 16; Lines 11 to 13, Paragraph [0062]; See also Fig. 17 and Paragraph [0063] that further describe operation 1615; Paragraph [0064] and Fig. 18 that further describe operation 1700 of Fig. 17.}. Each of the at least one input stack is associated with an input instruction of the first instruction. {Paragraph [0059]; lines 4 to 7, Paragraph [0062].} Each input stack represents the state of an operand stack associated with an input instruction upon execution of the input instruction. {See Paragraphs [083], [089], [0131], [0135], [0141] for examples of operation 1615. See also Paragraphs [0011], [0012] for a background discussion of operand stacks}.

In the converting, the first instruction is converted to a second instruction configured to operate on at least one operand of a second type {1625, Fig. 16; lines 13, 14, Paragraph [0062]; See also, Fig. 19 and Paragraph [0065] that further describe operation 1625.}. The second type is smaller than the first type and the converting is based at least in part on the relative size of the first type and the second type, wherein the second instruction is different from the first instruction {Paragraph [0065]; See also Paragraphs [083], [090], [0131], [0135], [0141] for examples of operation 1625.}.

In the matching, the second type is matched with an operand type of at least one operand in the at least one input stack associated with the second instruction {1630, Fig. 16; lines 14 to 16, Paragraph [0062]; See also, Fig. 20 and Paragraph [0067] that further describe operation 1630; Fig. 21 and Paragraph [0068] that describe operation 2025 in Fig. 20; and Fig. 22 and Paragraph [0069] that further describe operation 2105 in Fig. 21}. The matching comprises changing

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

the type of instructions in a chain of instructions to equal the second type if the operand type is less than the second type {Paragraph [0068]}. The chain is bounded by the second instruction and a third instruction that is the source of the at least one operand {Paragraph [0068]}. {See also paragraphs [084], [091], [106] to [0121], [132], [133] and [137] to [140] for examples of operation 1630.}.

CLAIM 78

With respect to Claim 78, a smart card {525, Fig. 5} includes a microcontroller {Paragraph [0015] configured to execute a virtual machine {Paragraph [0017]}. The virtual machine is capable of executing a software application {520, Fig. 5} comprising a plurality of previously optimized instructions. The instructions are optimized by a method including validating, converting and matching operations.

In the validating, at least one input stack associated with a first instruction configured to operate on at least one operand of a first type is validated {1615, Fig. 16; Lines 11 to 13, Paragraph [0062]; See also Fig. 17 and Paragraph [0063] that further describe operation 1615; Paragraph [0064] and Fig. 18 that further describe operation 1700 of Fig. 17.}. Each of the at least one input stack is associated with an input instruction of the first instruction. {Paragraph [0059]; lines 4 to 7, Paragraph [0062].} Each input stack represents the state of an operand stack associated with an input instruction upon execution of the input instruction. {See Paragraphs [083], [089], [0131], [0135], [0141] for examples of operation 1615. See also Paragraphs [0011], [0012] for a background discussion of operand stacks}.

In the converting, the first instruction is converted to a second instruction configured to operate on at least one

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

operand of a second type {1625, Fig. 16; lines 13, 14, Paragraph [0062]; See also, Fig. 19 and Paragraph [0065] that further describe operation 1625.}. The second type is smaller than the first type and the converting is based at least in part on the relative size of the first type and the second type, wherein the second instruction is different from the first instruction {Paragraph [0065]; See also Paragraphs [083], [090], [0131], [0135], [0141] for examples of operation 1625.}.

In the matching, the second type is matched with an operand type of at least one operand in the at least one input stack associated with the second instruction {1630, Fig. 16; lines 14 to 16, Paragraph [0062]; See also, Fig. 20 and Paragraph [0067] that further describe operation 1630; Fig. 21 and Paragraph [0068] that describe operation 2025 in Fig. 20; and Fig. 22 and Paragraph [0069] that further describe operation 2105 in Fig. 21}. The matching comprises changing the type of instructions in a chain of instructions to equal the second type if the operand type is less than the second type {Paragraph [0068]}. The chain is bounded by the second instruction and a third instruction that is the source of the at least one operand {Paragraph [0068]}. {See also paragraphs [084], [091], [106] to [0121], [132], [133] and [137] to [140] for examples of operation 1630.}.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

1. Whether Claims 1 to 10, 14 to 29, 33 to 48, 52, 53 to 67, 71 to 78 are unpatentable under 35 U.S.C. 103(a) over U.S. Patent No. 5,740,441, hereinafter referred to as Yellin, in view of U.S. Patent No. 6,308,317, hereinafter referred to as Wilkinson?

2. Whether Claims 11 to 13, 30 to 32, 49 to 51, 68 to 70, are unpatentable under 35 U.S.C. 103(a) over U.S. Patent No. 5,740,441, hereinafter referred to as Yellin, in view of U.S. Patent No. 6,308,317, hereinafter referred to as Wilkinson?

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

ARGUMENT

1. CLAIMS 1 TO 10, 14 TO 29, 33 TO 48, 52 TO 67, 71 TO 78 ARE PATENTABLE FOR MULTIPLE REASONS.

Claims 1 to 10, 14 to 29, 33 to 48, 52 to 67, 71 to 78 stand rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent No. 5,740,441, hereinafter referred to as Yellin, in view of U.S. Patent No. 6,308,317, hereinafter referred to as Wilkinson. Claims 1, 20, 39, 58, 77 and 78 are independent claims. Herein, Claim 1 is discussed. Each of Claims 20, 39, 58, 77 and 78 include similar limitations to Claim 1 or are narrower in scope than Claim 1 so that the remarks with respect to Claim 1 are applicable to each of the independent claims.

The rejection is deficient on multiple levels. The rejection has failed to consider the claims as a whole and as a result has not even alleged that one of the claim limitations is suggested by the combination of references. In fact, the rejection on its face demonstrates that the combination teaches away from Appellant's invention as recited in these claims.

The rejection ignores the requirements of the MPEP that each reference be considered as a whole. Pointing out the failure to comply with the requirements of the MPEP with respect to an obviousness analysis is not attacking the reference individually. The combination changes the principles of operation of the primary reference and so according to the MPEP the combination does not establish a prima facie obviousness rejection. Also, the rejection mischaracterizes the teachings of the primary reference, Yellin, and so fails to consider the reference as a whole.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

1.A. THE COMBINATION OF REFERENCES FAILS TO RENDER THE INVENTION OBVIOUS

With respect to an obviousness rejection, all words in a claim must be considered in judging the patentability of that claim against the prior art. MPEP § 2143.03, 8th Ed., Rev. 6, p. 2100-142 (Sept. 2007). This is a fundamental aspect in the requirement that **"THE CLAIMED INVENTION AS A WHOLE MUST BE CONSIDERED."** MPEP § 2141.02 I., 8th Ed., Rev. 6, p. 2100-123 (Sept. 2007). The rejection on its face demonstrates that all the words in the claim have not been considered and when all the words are considered, the rejection demonstrates that the combination teaches away from the rejection.

Claim 1 recites in part:

> matching said second type with an operand type of at least one operand in said at least one input stack associated with said second instruction

Thus, this portion of the claim requires matching and then recites in more detail what the matching includes, i.e.,

> said matching comprising **changing the type of instructions in a chain of instructions to equal said second type if said operand type is less than said second type**, (Emphasis added.)

The claim also expressly defines the extent of the chain:

> said chain bounded by said second instruction and a third instruction that is the source of said at least one operand

Thus, Claim 1 expressly recites that the type of instruction in the chain is changed to equal the second type if operand type is less than the second type and recites the precise extent of the chain.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

The claim language "if the operand type is less than the second type" means that the test is whether the operand type is smaller than the second type. When the operand type is less than the second type, i.e., smaller than the second type, the claim directs that the type of instruction is changed to be equal to the second type. Thus, the change is from a smaller type (less than the second type) to a larger type (the second type) according to the plain meaning of the claim.

The MPEP requires that the words of a claim must be given their "plain meaning" unless the plain meaning is inconsistent with the specification. MPEP § 2111.01, 8th Ed., Rev. 6, pg. 2100-38, (Sept. 2007). This interpretation is consistent with the specification, which stated "This method is used to recursively follow the chain of operand creation to change the origin of a smaller type and all of its subsequent instructions to a larger type." {Paragraph [0068]}.

The rejection noted the failure of Yellin to address the matching limitation by stating:

> nor does Yellin disclose that the above matching within a bounded chain includes changing the type of instructions in a chain of instructions to equal said second type *if said operand type is less than said second type.* (Emphasis in original.)

Final Office Action, March 27, 2008, pg 7, lines 9 to 11.

After noting this failure of Yellin, the rejection continued:

> Analogous to Yellin's approach as to modifying stack runtime Java bytecodes in order to accommodate an executing platform receiving/using data coming from a platform with higher architecture base (see Yellin: Background of the invention, col. 1), Wilkinson discloses modifying stack operands from a larger base platform to operands of lesser size that would fit the target platform (see Fig. 10-11; col. 11, lines 4-48), hence disclose a form of conversion from a word operand to a byte size operand of a lesser size than the word-based type operand. Based on the common endeavor by Yellin or Wilkinson to

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

> alleviate extraneous security and runtime resources of a given lower platform (e.g. receiving a JVM application destined for larger microcomputer into integrated circuit's microcontroller in portable devices) when loading operands for each instruction in Java method prior to runtime, it would have been obvious for one skill in the art at the time the invention was made to provide the operand type replacement as approached by Wilkinson so to support Yellin's loading process by way of converting at the receiving platform a larger type operand to a smaller size operand. (Emphasis Added.)

Final Office Action, March 27, 2008, pg 7, line 17 to pg. 8, line 7.

The rejection alleges only that the combination of references teaches converting based on the size of the operand from a large size to a smaller size. Assuming arguendo that the alleged teaching is correct, the matching recited in these claims, as quoted above, does not recite converting from a larger to a smaller, but in fact from a smaller to a larger. Thus, the rejection itself shows that the combination teaches away from the invention.

Therefore, Claim 1 distinguishes over the combination of references and is in condition for allowance.

Similarly, based upon these remarks, each of Claims 2 to 10, 14, 15, 20 to 29, 33, 34, 39 to 48, 52, 53, 58 to 67, 71, 72, and 77 to 78 is in condition for allowance.

The above is totally sufficient to overcome the obviousness rejection. Nevertheless, to avoid waiver, Appellant points out several additional reasons why the obviousness rejection is not well founded.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

1.B. A PRIMA FACIE OBVIOUSNESS REJECTION HAS NOT BEEN MADE

1.B.1. The Combination Of References Changes The Principles Of The Primary Reference

It is a basic tenet of an obviousness rejection that the references must be considered as a whole. "A prior art reference must be considered in its entirety." MPEP § 2141.02, VI., 8th Ed., Rev. 6, pg. 2100-126 (September 2007). The MPEP is unambiguous that in construing Yellin, Yellin must be considered in its entirety.

Therefore, even though there is a combination of references, each reference must support the interpretation attributed to that reference in the rejection. One way used in the MPEP to determine whether this requirement is satisfied with respect to a combination of references is:

> If the proposed modification or combination of the prior art would change the principle of operation of the prior art invention being modified, then the teachings of the references are not sufficient to render the claims *prima facie* obvious.

MPEP § 2143.01 VI, 8th Ed., Rev. 6, pg. 2100-141, (Sept. 2007).

The rejection is replete with mischaracterizations of Yellin that change the principles of operation of Yellin. For example,

> Analogous to Yellin's approach as to modifying stack runtime Java bytecodes in order to accommodate an executing platform receiving/using data coming from a platform with higher architecture base

Final Office Action, March 27, 2008, pg 7, line 17 to 19.

Reading this would make one think that Yellin is concerned with modifying a bytecode program to accommodate different platforms. This is completely false. Yellin does not modify

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

any bytecode program, but rather simply verifies an existing bytecode program. If the verification fails, use of the bytecode program is blocked. Yellin does not teach porting or modifying the bytecode program in any way.

Specifically, Yellin is not concerned with (1) modifying bytecodes, (2) porting code from one computer platform to a different computer platform; or (3) arithmetic overflows. Yellin succinctly describes:

> The present invention **verifies the integrity of computer programs written in a bytecode language**, commercialized as the JAVA bytecode language, ...
> The present invention provides a verifier tool and method for identifying, prior to execution of a bytecode program, any instruction sequence that attempts to process data of the wrong type for such a bytecode or if the execution of any bytecode instructions in the specified program would cause underflow or overflow of the operand stack, and to prevent the use of such a program. (All Emphasis Added)

Yellin, Col. 1, line 57 to Col. 2, line 7.

One of skill in the art understands that a bytecode program is computer platform independent and so there is no need for the conjectured porting in the rejection. Yellin starts with a bytecode program that is computer platform independent and teaches a verifier that verifies that bytecode program, but in the verification process, the bytecode program is not modified. The reason is that Yellin does not know, for example, whether the problem is simply a programming error or malicious code and so simply blocks use of the program if the verification fails.

Specifically, Yellin describes that if the program tries to process data of the wrong type for a particular bytecode, or if execution of a bytecode program would underflow or overflow the operand stack, the use the program is prevented as quoted above. The remainder of Yellin describes how the features of

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

the verifier are implemented and not any process for optimization of porting between computer platforms.

Thus, Yellin, taken as a whole, is clear and unambiguous as to what is done (bytecode program is verified); when it is done (prior to execution); and the action taken if the verification is unsuccessful (use of the program is prevented.) Yellin simply does not suggest any modification to the bytecode program and rather expressly taught that use of the entire bytecode program was prevented when verification failed.

Consequently, any change to Yellin that permits execution when a problem is found, as suggested in the rejection, changes the principles of operation of Yellin. If a problem with an instruction is detected by Yellin and is corrected using Wilkinson to optimize the instruction, for example, malicious code could be enabled and this is exactly what Yellin is trying to avoid. Specifically, Yellin stated:

> . . . . Even worse, a bytecode program might attempt to create object references (e.g., by loading a computed number into the operand stack and then attempting to use the computed number as an object handle) and to thereby breach the security and/or integrity of the user's computer.

Yellin, Col. 5, lines 60 to 64.

Thus, if Yellin were modified to optimize such a statement as asserted in the rejection, Yellin would no longer prevent breach of security and/or integrity of the user's program, which is a change in the principle of operation. The modification would change the principles of operation of Yellin that explicitly described that when a problem was detected, program use was prevented. Thus, the references are not sufficient to render the claims prima facie obvious according to the MPEP, as quoted above.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

1.B.2. The Rejection Is Based On Mischaracterizations Of Yellin

Yellin succinctly described the purpose of the bytecode verifier as:

> Use of the bytecode verifier 120 in accordance with the present invention enables verification of a bytecode program's integrity and allows the use of an interpreter 122 which does not execute the usual stack monitoring instructions during program execution, thereby greatly accelerating the program interpretation process.

Yellin, Col. 5, line 54 to Col. 6, line 3.

Yellin addresses verification only and does not suggest or teach converting any instruction. Yellin steps through the code and if an illegal condition is found simply aborts the verification process. Each of Figs. 4A to 4C of Yellin show that if an improper condition is found the verification is aborted. There is no operation of Yellin that teaches or suggests doing anything to one instruction to obtain another instruction as recited in Claim 1 and as asserted in the rejection.

Nevertheless, the rejection contorts and mischaracterizes these explicit teachings and thereby fails to consider Yellin as a whole. The rejection takes portions of Yellin that describe issues with the prior art and why the bytecode program is preferable and attributes the prior art problems to processes performed by the bytecode program verifier. This interpretation not only mischaracterizes the reference, but also goes against the understanding of those of skill in the art of the characteristics of a bytecode program as described by Yellin.

As a first example, the rejection stated as a basis for the combination of references:

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

> However, Yellin discloses adding data and rearranging stack calling structure relative to discrepancies of first and second type (see col. 1, lines 60-67; col. 20, lines 24-35; col. 21, lines 12-37; step 394, Fig. 4B) (Emphasis in original)

Final Office Action, March 27, 2008, pg 7, lines 11 to 13.

Examination below of each of the above cited sections in Yellin demonstrates that Yellin has been misinterpreted and does not teach or suggest any "adding data and rearranging stack calling structure relative to discrepancies of first and second type." As shown below, when each of these sections are considered in the context of Yellin as a whole, none of the cited sections support this interpretation.

Col. 1 lines 60 to 67 of Yellin stated:

> All the available source code bytecodes in the language either (A) are stack data consuming bytecodes that have associated data type restrictions as to the types of data that can be processed by each such bytecode, (B) do not utilize stack data but affect the stack by either adding data of known data type to the stack or by removing data from the stack without regard to data type, or (C) neither use stack data nor add data to the stack.

This section of Yellin describes the bytecodes of the JAVA bytecode language and as noted above, it is well known that such bytecodes are computer platform independent. This section provides no support for the assertion that Yellin is "adding data and rearranging stack calling structure relative to discrepancies of first and second type," as asserted in the above quoted portion of the rejection.

Col. 20, lines 24 to 35 of Yellin in context of lines 23 to 42 recite:

```
/* Update jsr bit vector array */
If the current instruction is in a subroutine that is the
target of a jsr
{
For each level of jsr applicable to the current instruction
{
```

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

```
Update corresponding jsr bit vector to indicate register(s)
accessed or modified by the current instruction
/* Set of "marked" registers can only be increased, not
decreased */
 }
 }
 }
/* Update all affected SnapShots and changed bits */
Determine set of all successor instructions, including:
(A) the next instruction if the current instruction is not an
unconditional goto, a return, or a throw,
(B) the target of a conditional or unconditional branch,
(C) all exception handlers for this instruction,
(D) when the current instruction is a return instruction, the
successor instructions are the instructions immediately
following all jsr's that target the called subroutine.

If the program can "fall off" the last instruction
{ Set VerificationSuccess to False
 Return with Abort return code value }
/*
```

This section of pseudocode describes the **bytecode verifier** that is the subject of Yellin's invention and not the bytecode program being verified. Specifically, with respect to the bytecode verifier, Yellin is concerned with updating bits associated with "a "jsr" bit vector array 306 that stores zero or more bit vectors associated with the zero or more subroutine calls required to reach the instruction currently being processed" (Yellin, Col. 6, lines 19 to 22) and updating a snapshot array of the verifier. The Examiner ignores that Yellin is describing how to make and use a bytecode program verifier and is not concerned with modifying the bytecode program. Yellin is describing structure in the byte code verifier that is used in verification of the bytecode program. Thus, this section also fails to support the assertion in the rejection.

In particular, this portion of Yellin has nothing to do with modification to the bytecode program as asserted in the above quoted portion of the rejection. The program code for the bytecode verifier teaches or suggests nothing with respect to "adding data and rearranging stack calling structure

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

relative to discrepancies of first and second type" in the bytecode program being verified.

Similarly, Col. 21, lines 12 to 37 of Yellin are pseudocode for elements of the bytecode verifier used to verify the bytecode program and not the bytecode program being verified. Changing a snapshot in the instruction verifier as taught in step 394 of Fig. 4B again teaches what the verifier does in verification process and fails to teach or suggest anything with respect to "adding data and rearranging stack calling structure relative to discrepancies of first and second type" in the bytecode program being verified by Yellin.

Thus, as previously stated, the rejection mischaracterizes the teaching of Yellin and relies upon this mischaracterization to assert that Appellant's claims are obvious. This demonstrates that Yellin fails to provide any support for the interpretation in the rejection, which formed the basis of the rationale for the combination of references.

The mischaracterizations of Yellin continue. The rejection continues:

> hence has suggested modifying the runtime instruction as verified by the stack to accommodate data being ported from a larger size operand --or higher platform-- to a smaller size operand - or lower platform (see col. 5, lines 14-64; *multiple computer platforms, underlying instruction sets*- col. *1* , lines 7- 14).

Final Office Action, March 27, 2008, pg 7, line 14 to line 17.

The conclusion reached is directly contradicted by the cited section of Yellin and can only be based on Appellant's claim language, which is an impermissible form of analysis. "[I]mpermissible hindsight must be avoided and the legal conclusion must be reached on the basis of the facts gleaned from the prior art." MPEP § 2142, 8th Ed., Rev. 6, pg. 2100-127 (September 2007).

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

Specifically, Col. 5, lines 14 to 31 of Yellin described the prior art problems with multiple different computer architectures. Col. 5, lines 32 to 34 describe the solution to that problem, a single bytecode program.

Thus, contrary to any need to port data, Yellin expressly taught that a single bytecode version of the program could be used. Yellin expressly taught that the use of a bytecode program as in Yellin negated the need to develop a program for each different architecture. Moreover, one of skill in the art would recognize this as one of the principle advantages of the JAVA programming language cited by Yellin. Thus, the rejection not only contradicts Yellin but also goes against the common understanding of what a bytecode program is.

Lines 35 to 42 in Col. 5 of Yellin describe the operation of the bytecode verifier, i.e., and have nothing to do with the bytecode program being modified as asserted in the rejection. These lines teach or suggest nothing about porting data as alleged in the rejection. Lines 43 to 53 describe prior art problems with having to monitor for stack overflows and underflows during execution of a bytecode program. Since the bytecode verifier checks for such problems and blocks execution if the problems are encountered, this portion of Yellin is describing an advantage of the byte code verifier and has nothing to do with the assertions in the above quoted portion of the rejection.

Similarly, lines 54 to 64 describe prior art problems that are stopped by use of the bytecode program verifier and so yet again has nothing to do with the interpretation in the rejection.

Thus, Appellant has demonstrated that the portion of the rejection that is used as a basis for combining Yellin with Wilkinson is erroneous and totally mischaracterizes and misrepresents the teaching of Yellin. This is not an attack on Yellin individually as asserted by the Examiner, but rather a

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

demonstration that Yellin does not support the interpretation asserted in the rejection. While *KSR International Co. v. Teleflex Inc.*, 550 U.S. ___, ___, 82 USPQ2d 1385, 1397 (2007 may have changed the standard for combination of references, KSR cannot be used to justify the wholesale misinterpretation and mischaracterization of a reference.

Thus, Appellant has demonstrated that the rejection failed to consider Yellin as a whole, and as a result put forth a rationale for modifying Yellin based on an erroneous interpretation of Yellin. Thus, the references are not sufficient to render the claims prima facie obvious according to the MPEP. Appellant notes that any one of the above showings is sufficient to overcome the obviousness rejection of each of Claims 1 to 10, 14 to 29, 33 to 48, 52, 53 to 67, and 71 to 78.

2. CLAIMS 11 to 13, 30 to 32, 49 to 51, 68 to 70 ARE PATENTABLE FOR MULTIPLE REASONS.

Claims 11 to 13, 30 to 32, 49 to 51, 68 to 70 stand rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent No. 5,740,441, hereinafter referred to as Yellin, in view of U.S. Patent No. 6,308,317, hereinafter referred to as Wilkinson.

Claim 11 depends from 1 and distinguishes over the combination of references for reasons in addition to Claim 1. Claims 12 to 13 depend from Claim 11. Claims 30, 49, and 68 include limitations similar to those in Claim 11. Thus, Claim 11 is used as an example for this set of claims.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

2.A. THE COMBINATION OF REFERENCES FAILS TO RENDER THE INVENTION OBVIOUS

With respect to an obviousness rejection, all words in a claim must be considered in judging the patentability of that claim against the prior art. MPEP § 2143.03, 8th Ed., Rev. 6, p. 2100-142 (Sept. 2007). This is a fundamental aspect in the requirement that **"THE CLAIMED INVENTION AS A WHOLE MUST BE CONSIDERED."** MPEP § 2141.02 I., 8th Ed., Rev. 6, p. 2100-123 (Sept. 2007). The rejection on its face demonstrates that all the words in the claim have not been considered and when all the words are considered, the rejection demonstrates that the combination teaches away from the rejection.

For example, with respect to Claim 11, Claim 11 recites three different setting processes that further define the converting of Claim 1. The third setting process recites:

> setting said second type to a type that is larger than said smallest type (1) if said smallest type is greater than said type of said operand, (2) if said operand has potential overflow, (3) if said second instruction is sensitive to overflow and (4) if said second type is less than said first type. (Numbers inserted for discussion)

Thus, this part of Claim 11 requires that four different conditions be true for setting the second type to a type that is larger than the smallest type. As noted above, the MPEP requires that the claims be considered as a whole.

Claim 11 is directed at arithmetic expression optimization process, thus the instructions, operands etc. recited in the claims are interpreted with respect to such an optimization. In claim 11, the claim preamble, when read in the context of the entire claim, recites limitations of the claim in that the preamble defines the context of elements used in the process. Thus, according to the MPEP "the claim preamble should be

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

construed as if in the balance of the claim." MPEP § 2111.02, 8th Ed., Rev. 6, p. 2100-41 (Sept. 2007).

Moreover, such an interpretation is consistent with the requirement in an obviousness rejection that inherent properties of the Claim that are disclosed in the specification must be considered in that "as a whole" analysis of the claim. MPEP § 2141.02, V., 8th Ed., Rev. 6, p. 2100-125 (Sept. 2007). The examples, starting at Paragraph [0079,] demonstrate that the overflow considerations are with respect to an arithmetic expression. The inherent properties of the elements in Claim 11, as disclosed in the specification, are that the elements are associated with an arithmetic expression being optimized. Thus, when the claims are interpreted as a whole in view of the various MPEP requirements, Claim 11 is not directed as some abstract environment in general where overflow occurs, but rather overflow with respect to parts of a particular type of expression.

Overflow associated with an arithmetic expression is interpreted by those of skill as being different from a stack overflow or underflow as used by Yellin. In particular, Yellin expressly described what these stack overflows mean to one of skill in the art:

> monitor the operand stack for overflows (i.e., adding more data to the stack than the stack can store) and underflows (i.e., attempting to pop data off the stack when the stack is empty).

Yellin, Col. 5, lines 45 to 48.

If the operand stack has insufficient room to store the data to be pushed onto the stack by the current instruction (472), that is called a stack overflow

Yellin, Col. 9, lines 65 to 67.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

Thus, the underflows and overflows, as taught by Yellin are directed to storing and retrieving data on a stack. Wilkinson was electronically searched for "overflow" and "underflow" and the words were not found. Thus, the teaching of the combination of references with respect to "overflow" is limited to one very specific overflow, a stack overflow, which is defined as insufficient room to store the data on the stack.

The rejection treats all overflows as equivalent, which is error. Appellant has already demonstrated that when Claim 11 is interpreted as a whole, and when the combinations of references are interpreted in view of the level of skill in the art, as established by those references, the use of "overflow" in Yellin is different from the "overflow" in Claim 11 and so the combination of references fails to teach or suggest anything with respect to Claim 11 and so cannot render Claim 11 obvious.

However, there is even further proof, of an improper claim interpretation. The specification defines:

> **[0037]** Since the actual values used in an arithmetic operation are not known at optimization time, the optimization must assume the worst case value for each operand. The worst case value for an operand is determined based upon the input operand type. A small-type operation can have results that require large-type representation or overflow into a larger type. Thus, according to embodiments of the present invention, arithmetic operators are categorized into operators affected by overflow and operators with the potential to create overflow. For the purposes of this disclosure, overflow comprises the underflow of negative values. The result of a small-type operation is said to carry potential overflow if the operator used to create the result belongs to the group of operators with the potential to create overflow into a large-type representation. Intermediate values are allowed to carry potential overflow as long as the intermediate value is not used as an operand for an operator belonging to the group of operators affected by overflow. (Emphasis Added.)

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

Thus, the specification provides a definition of overflow that is consistent with the interpretation of the claim limitation taken as a whole as discussed above. Moreover, the MPEP also directs:

> Where an explicit definition is provided by the applicant for a term, that definition will control interpretation of the term as it is used in the claim.

MPEP § 2106, 8th Ed., Rev. 6, p. 2100-7 (Sept. 2007).

Thus, "overflow" in Claim 11, when interpreted as required by the MPEP, has nothing to do with stack storage availability as defined by Yellin. The rejection totally ignores that Yellin is directed at stack overflow and simply treats all uses of overflow as equivalent.

As previously noted the rejection also makes statements about Yellin that have nothing to do with Yellin including the statements about overflow. The rejection also reduces the express claim limitations to a gist, which is an improper form of analysis. MPEP § 2141.02 II., 8th Ed., Rev. 6, p. 2100-124 (Sept. 2007).

With respect to the above quoted claim limitation of Claim 11, the rejection stated:

> setting said second type to a type that is larger than said smallest type if said smallest type is greater than said type of said operand, if said operand has potential overflow, if said second instruction is sensitive to overflow and if said second type is less than said first type (see rationale of claim 1; according to which any type if predicted to overflow beyond a smaller size will be replaced, i.e. overflow analysis as by Yellin inherently teaching setting a smallest type to a larger type when said smallest type is itself greater than type of any operand potentially risking being overflowed)

There is no showing of any teaching or suggestion of a setting process only when four different conditions are met as

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

recited in the claim. Instead, the four distinct conditions are lumped together and treated as a single requirement. Also, there is no inherent teaching in Yellin with respect to an instruction overflow that is part of an arithmetic expression and this further demonstrates a failure to consider Yellin as a whole. As demonstrated above, Yellin is directed at stack overflow, which was defined as no storage space availability on the stack. Again, the rejection uses Appellant's disclosure and claims as a roadmap, which the MPEP specifically indicates is impermissible as quoted above, and lumps them together, which is also impermissible.

Therefore, Claim 11 distinguishes over the combination of references in addition to the reasons given above for Claim 1, and is in condition for allowance.

Similarly, based upon these remarks, each of Claims 12 to 13, 30 to 32, 49 to 51, 68 to 70 is in condition for allowance.

CONCLUSION

For the reasons above, all appealed claims, i.e., Claims 1 to 78, are allowable. Appellant respectfully requests the Board of Patent Appeals and Interferences to reverse the Examiner's rejections under U.S.C. § 103(a) of these claims.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

CLAIMS APPENDIX

1. (Previously Presented) A method for arithmetic expression optimization, comprising:

validating at least one input stack associated with a first instruction configured to operate on at least one operand of a first type, each of said at least one input stack associated with an input instruction of said first instruction, each input stack representing the state of an operand stack associated with an input instruction upon execution of said input instruction;

converting said first instruction to a second instruction configured to operate on at least one operand of a second type, said second type smaller than said first type, said converting based at least in part on the relative size of said first type and said second type, wherein said second instruction is different from said first instruction; and

matching said second type with an operand type of at least one operand in said at least one input stack associated with said second instruction, said matching comprising changing the type of instructions in a chain of instructions to equal said second type if said operand type is less than said second type, said chain bounded by said second instruction and a third instruction that is the source of said at least one operand.

2. (Original) The method of claim 1 wherein said first instruction is arithmetic.

3. (Original) The method of claim 1 wherein said first instruction comprises a non-arithmetic, type-sensitive instruction.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

4. (Previously Presented) The method of claim 1, further comprising repeating said validating, said converting and said matching for instructions that comprise a program.

5. (Original) The method of claim 1, further comprising linking each instruction to input instructions in all control paths.

6. (Original) The method of claim 1 wherein

said first instruction is defined for a first processor having a first base; and

said second instruction is defined for a second processor having a second base.

7. (Original) The method of claim 6 wherein

said first processor comprises a Java™ Virtual Machine; and

said second processor comprises a Java Card™ Virtual Machine.

8. (Original) The method of claim 6 wherein

said first processor comprises a 32-bit processor; and

said second processor comprises a resource-constrained 16-bit processor.

9. (Original) The method of claim 1 wherein

said at least one input stack comprises a plurality of input stacks, said plurality of input stacks further comprising a first input stack and a second input stack; and

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

said validating further comprises comparing operand types of corresponding entries in said first input stack and said second input stack.

10. (Original) The method of claim 9 wherein said comparing further comprises indicating an error if the types of at least said first at least one stack entry and said second at least one stack entry are not equivalent.

11. (Previously Presented) The method of claim 1 wherein said converting further comprises:

setting said second type to a smallest type;

setting said second type to the type of an operand in said at least one input stack if said smallest type is less than said type of said operand; and

setting said second type to a type that is larger than said smallest type if said smallest type is greater than said type of said operand, if said operand has potential overflow, if said second instruction is sensitive to overflow and if said second type is less than said first type.

12. (Original) The method of claim 11 wherein said smallest type is the smallest type supported by a target processor.

13. (Previously Presented) The method of claim 11 wherein said smallest type is the smallest type determined during a previous pass of said converting.

14. (Original) The method of claim 1 wherein

said third instruction is not a source instruction; and

said changing further comprises:

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

recursively examining input instructions until said third instruction is obtained; and

setting the type of said third instruction to equal said second type.

15. (Original) The method of claim 1 wherein

said third instruction comprises a source instruction; and

said changing further comprises:

recursively examining input instructions until said third instruction is obtained;

setting the type of said third instruction to equal said second type; and

repeating said changing for each input instruction of said third instruction.

16. (Original) The method of claim 1, further comprising recording conversion results, said recording comprising:

determining potential overflow associated with said second instruction; and

generating an output stack based at least in part on execution of said second instruction.

17. (Original) The method of claim 16 wherein said determining further comprises:

indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if said second instruction creates potential overflow; and

indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if said second instruction does not create

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

potential overflow, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow.

18. (Original) The method of claim 17 wherein said determining further comprises:

indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if overflow is possible based at least in part on the type of said second instruction and the relationship between said first type and said second type; and

indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if overflow is not possible based at least in part on the type of said second instruction and the relationship between said first type and said second type, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow.

19. (Original) The method of claim 16 wherein said generating further comprises:

creating a new output stack based at least in part on one of said at least one input stack;

updating said new output stack based at least in part on operation of said second instruction; and

indicating another instruction conversion pass is required if said new stack does not equal a previous output stack.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

20. (Previously Presented) A method for arithmetic expression optimization, comprising:

step for validating at least one input stack associated with a first instruction configured to operate on at least one operand of a first type, each of said at least one input stack associated with an input instruction of said first instruction, each input stack representing the state of an operand stack associated with an input instruction upon execution of said input instruction;

step for converting said first instruction to a second instruction configured to operate on at least one operand of a second type, said second type smaller than said first type, said converting based at least in part on the relative size of said first type and said second type, wherein said second instruction is different from said first instruction; and

step for matching said second type with an operand type of at least one operand in said at least one input stack associated with said second instruction, said matching comprising changing the type of instructions in a chain of instructions to equal said second type if said operand type is less than said second type, said chain bounded by said second instruction and a third instruction that is the source of said at least one operand.

21. (Original) The method of claim 20 wherein said first instruction is arithmetic.

22. (Original) The method of claim 20 wherein said first instruction comprises a non-arithmetic, type-sensitive instruction.

23. (Previously Presented) The method of claim 20, further comprising step for repeating said validating, said

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

converting and said matching for instructions that comprise a program.

24. (Original) The method of claim 20, further comprising step for linking each instruction to input instructions in all control paths.

25. (Original) The method of claim 20 wherein

said first instruction is defined for a first processor having a first base; and

said second instruction is defined for a second processor having a second base.

26. (Original) The method of claim 25 wherein

said first processor comprises a Java™ Virtual Machine; and

said second processor comprises a Java Card™ Virtual Machine.

27. (Original) The method of claim 25 wherein

said first processor comprises a 32-bit processor; and

said second processor comprises a resource-constrained 16-bit processor.

28. (Original) The method of claim 20 wherein

said at least one input stack comprises a plurality of input stacks, said plurality of input stacks further comprising a first input stack and a second input stack; and

said validating further comprises comparing operand types of corresponding entries in said first input stack and said second input stack.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

29. (Original) The method of claim 28 wherein said step for comparing further comprises step for indicating an error if the types of at least said first at least one stack entry and said second at least one stack entry are not equivalent.

30. (Previously Presented) The method of claim 20 wherein said converting further comprises:

step for setting said second type to a smallest type;

step for setting said second type to the type of an operand in said at least one input stack if said smallest type is less than said type of said operand; and

step for setting said second type to a type that is larger than said smallest type if said smallest type is greater than said type of said operand, if said operand has potential overflow, if said second instruction is sensitive to overflow and if said second type is less than said first type.

31. (Original) The method of claim 30 wherein said smallest type is the smallest type supported by a target processor.

32. (Previously Presented) The method of claim 30 wherein said smallest type is the smallest type determined during a previous pass of said converting.

33. (Original) The method of claim 20 wherein

said third instruction is not a source instruction; and

said step for changing further comprises:

step for recursively examining input instructions until said third instruction is obtained; and

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

step for setting the type of said third instruction to equal said second type.

34. (Original) The method of claim 20 wherein said third instruction comprises a source instruction; and

said step for changing further comprises:

step for recursively examining input instructions until said third instruction is obtained;

step for setting the type of said third instruction to equal said second type; and

step for repeating said changing for each input instruction of said third instruction.

35. (Original) The method of claim 20, further comprising step for recording conversion results, said recording comprising:

step for determining potential overflow associated with said second instruction; and

step for generating an output stack based at least in part on execution of said second instruction.

36. (Original) The method of claim 35 wherein said step for determining further comprises:

step for indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if said second instruction creates potential overflow; and

step for indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if said second instruction does not

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

create potential overflow, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow.

37. (Original) The method of claim 36 wherein said step for determining further comprises:

step for indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if overflow is possible based at least in part on the type of said second instruction and the relationship between said first type and said second type; and

step for indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if overflow is not possible based at least in part on the type of said second instruction and the relationship between said first type and said second type, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow.

38. (Original) The method of claim 35 wherein said step for generating further comprises:

step for creating a new output stack based at least in part on one of said at least one input stack;

step for updating said new output stack based at least in part on operation of said second instruction; and

step for indicating another instruction conversion pass is required if said new stack does not equal a previous output stack.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

39. (Previously Presented) A program storage device readable by a machine, embodying a program of instructions executable by the machine to perform a method for arithmetic expression optimization, the method comprising:

validating at least one input stack associated with a first instruction configured to operate on at least one operand of a first type, each of said at least one input stack associated with an input instruction of said first instruction, each input stack representing the state of an operand stack associated with an input instruction upon execution of said input instruction;

converting said first instruction to a second instruction configured to operate on at least one operand of a second type, said second type smaller than said first type, said converting based at least in part on the relative size of said first type and said second type, wherein said second instruction is different from said first instuction; and

matching said second type with an operand type of at least one operand in said at least one input stack associated with said second instruction, said matching comprising changing the type of instructions in a chain of instructions to equal said second type if said operand type is less than said second type, said chain bounded by said second instruction and a third instruction that is the source of said at least one operand.

40. (Original) The program storage device of claim 39 wherein said first instruction is arithmetic.

41. (Original) The program storage device of claim 39 wherein said first instruction comprises a non-arithmetic, type-sensitive instruction.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

42. (Previously Presented) The program storage device of claim 39, said method further comprising repeating said validating, said converting and said matching for instructions that comprise a program.

43. (Original) The program storage device of claim 39, said method further comprising linking each instruction to input instructions in all control paths.

44. (Original) The program storage device of claim 39 wherein

said first instruction is defined for a first processor having a first base; and

said second instruction is defined for a second processor having a second base.

45. (Original) The program storage device of claim 44 wherein

said first processor comprises a Java™ Virtual Machine; and

said second processor comprises a Java Card™ Virtual Machine.

46. (Original) The program storage device of claim 44 wherein

said first processor comprises a 32-bit processor; and

said second processor comprises a resource-constrained 16-bit processor.

47. (Original) The program storage device of claim 39 wherein

said at least one input stack comprises a plurality of input stacks, said plurality of input stacks further

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

comprising a first input stack and a second input stack; and

said validating further comprises comparing operand types of corresponding entries in said first input stack and said second input stack.

48. (Original) The program storage device of claim 47 wherein said comparing further comprises indicating an error if the types of at least said first at least one stack entry and said second at least one stack entry are not equivalent.

49. (Previously Presented) The program storage device of claim 39 wherein said converting further comprises:

setting said second type to a smallest type;

setting said second type to the type of an operand in said at least one input stack if said smallest type is less than said type of said operand; and

setting said second type to a type that is larger than said smallest type if said smallest type is greater than said type of said operand, if said operand has potential overflow, if said second instruction is sensitive to overflow and if said second type is less than said first type.

50. (Original) The program storage device of claim 49 wherein said smallest type is the smallest type supported by a target processor.

51. (Previously Presented) The program storage device of claim 49 wherein said smallest type is the smallest type determined during a previous pass of said converting.

52. (Original) The program storage device of claim 39 wherein

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

said third instruction is not a source instruction; and

said changing further comprises:

recursively examining input instructions until said third instruction is obtained; and

setting the type of said third instruction to equal said second type.

53. (Original) The program storage device of claim 39 wherein

said third instruction comprises a source instruction; and

said changing further comprises:

recursively examining input instructions until said third instruction is obtained;

setting the type of said third instruction to equal said second type; and

repeating said changing for each input instruction of said third instruction.

54. (Original) The program storage device of claim 39, further comprising recording conversion results, said recording comprising:

determining potential overflow associated with said second instruction; and

generating an output stack based at least in part on execution of said second instruction.

55. (Original) The program storage device of claim 54 wherein said determining further comprises:

indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

overflow, and if said second instruction creates potential overflow; and

indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if said second instruction does not create potential overflow, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow.

56. (Original) The program storage device of claim 55 wherein said determining further comprises:

indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if overflow is possible based at least in part on the type of said second instruction and the relationship between said first type and said second type; and

indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if overflow is not possible based at least in part on the type of said second instruction and the relationship between said first type and said second type, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow.

57. (Original) The program storage device of claim 54 wherein said generating further comprises:

creating a new output stack based at least in part on one of said at least one input stack;

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

updating said new output stack based at least in part on operation of said second instruction; and

indicating another instruction conversion pass is required if said new stack does not equal a previous output stack.

58. (Previously Presented) An apparatus for arithmetic expression optimization, comprising:

means for validating at least one input stack associated with a first instruction configured to operate on at least one operand of a first type, each of said at least one input stack associated with an input instruction of said first instruction, each input stack representing the state of an operand stack associated with an input instruction upon execution of said input instruction;

means for converting said first instruction to a second instruction configured to operate on at least one operand of a second type, said second type smaller than said first type, said converting based at least in part on the relative size of said first type and said second type, wherein said second instruction is different from said first instruction; and

means for matching said second type with an operand type of at least one operand in said at least one input stack associated with said second instruction, said matching comprising changing the type of instructions in a chain of instructions to equal said second type if said operand type is less than said second type, said chain bounded by said second instruction and a third instruction that is the source of said at least one operand.

59. (Original) The apparatus of claim 58 wherein said first instruction is arithmetic.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

60. (Original) The apparatus of claim 58 wherein said first instruction comprises a non-arithmetic, type-sensitive instruction.

61. (Previously Presented) The apparatus of claim 58, further comprising means for repeating said validating, said converting and said matching for instructions that comprise a program.

62. (Original) The apparatus of claim 58, further comprising means for linking each instruction to input instructions in all control paths.

63. (Original) The apparatus of claim 58 wherein
said first instruction is defined for a first processor having a first base; and
said second instruction is defined for a second processor having a second base.

64. (Original) The apparatus of claim 63 wherein
said first processor comprises a Java™ Virtual Machine; and
said second processor comprises a Java Card™ Virtual Machine.

65. (Original) The apparatus of claim 63 wherein
said first processor comprises a 32-bit processor; and
said second processor comprises a resource-constrained 16-bit processor.

66. (Original) The apparatus of claim 58 wherein
said at least one input stack comprises a plurality of input stacks, said plurality of input stacks further

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

comprising a first input stack and a second input stack; and

said means for validating further comprises means for comparing operand types of corresponding entries in said first input stack and said second input stack.

67. (Original) The apparatus of claim 66 wherein said means for comparing further comprises means for indicating an error if the types of at least said first at least one stack entry and said second at least one stack entry are not equivalent.

68. (Previously Presented) The apparatus of claim 58 wherein said means for converting further comprises:

means for setting said second type to a smallest type;

means for setting said second type to the type of an operand in said at least one input stack if said smallest type is less than said type of said operand; and

means for setting said second type to a type that is larger than said smallest type if said smallest type is greater than said type of said operand, if said operand has potential overflow, if said second instruction is sensitive to overflow and if said second type is less than said first type.

69. (Original) The apparatus of claim 68 wherein said smallest type is the smallest type supported by a target processor.

70. (Previously Presented) The apparatus of claim 68 wherein said smallest type is the smallest type determined during a previous pass of said converting.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

71. (Original) The apparatus of claim 58 wherein

said third instruction is not a source instruction; and

said means for changing further comprises:

means for recursively examining input instructions until said third instruction is obtained; and

means for setting the type of said third instruction to equal said second type.

72. (Original) The apparatus of claim 58 wherein

said third instruction comprises a source instruction; and

said means for changing further comprises:

means for recursively examining input instructions until said third instruction is obtained;

means for setting the type of said third instruction to equal said second type; and

means for repeating said changing for each input instruction of said third instruction.

73. (Original) The apparatus of claim 58, further means for comprising recording conversion results, said recording comprising:

means for determining potential overflow associated with said second instruction; and

means for generating an output stack based at least in part on execution of said second instruction.

74. (Original) The apparatus of claim 73 wherein said means for determining further comprises:

means for indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

potential overflow, and if said second instruction creates potential overflow; and

means for indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if said second instruction does not create potential overflow, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow.

75. (Original) The apparatus of claim 74 wherein said determining further comprises:

means for indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if overflow is possible based at least in part on the type of said second instruction and the relationship between said first type and said second type; and

means for indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if overflow is not possible based at least in part on the type of said second instruction and the relationship between said first type and said second type, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow.

76. (Original) The apparatus of claim 73 wherein said means for generating further comprises:

means for creating a new output stack based at least in part on one of said at least one input stack;

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

means for updating said new output stack based at least in part on operation of said second instruction; and

means for indicating another instruction conversion pass is required if said new stack does not equal a previous output stack.

77. (Previously Presented) A method of using an application software program including arithmetic expression optimization of at least one instruction targeted to a processor, the method comprising:

receiving the software program on said processor, said software program optimized according to a method comprising:

validating at least one input stack associated with a first instruction configured to operate on at least one operand of a first type, each of said at least one input stack associated with an input instruction of said first instruction, each input stack representing the state of an operand stack associated with an input instruction upon execution of said input instruction;

converting said first instruction to a second instruction configured to operate on at least one operand of a second type, said second type smaller than said first type, said converting based at least in part on the relative size of said first type and said second type, wherein said second instruction is different from said first instruction; and

matching said second type with an operand type of at least one operand in said at least one input stack associated with said second instruction, said matching comprising changing the type of instructions in a chain of instructions to equal said second type if said operand type is less than said second type,

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

said chain bounded by said second instruction and a third instruction that is the source of said at least one operand; and

executing said at least one instruction on said processor.

78. (Previously Presented) A smart card having a microcontroller embedded therein, said microcontroller configured to execute a virtual machine, the virtual machine capable of executing a software application comprising a plurality of previously optimized instructions, the instructions optimized by a method comprising:

validating at least one input stack associated with a first instruction configured to operate on at least one operand of a first type, each of said at least one input stack associated with an input instruction of said first instruction, each input stack representing the state of an operand stack associated with an input instruction upon execution of said input instruction;

converting said first instruction to a second instruction configured to operate on at least one operand of a second type, said second type smaller than said first type, said converting based at least in part on the relative size of said first type and said second type, wherein said second instruction is different from said first instruction; and

matching said second type with an operand type of at least one operand in said at least one input stack associated with said second instruction, said matching comprising changing the type of instructions in a chain of instructions to equal said second type if said operand type is less than said second type, said chain bounded by said second instruction and a third instruction that is the source of said at least one operand.

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

EVIDENCE APPENDIX

None

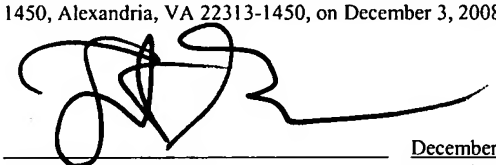
GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888

RELATED PROCEEDINGS APPENDIX

None

CERTIFICATE OF MAILING

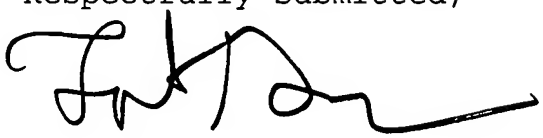
I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on December 3, 2008.



Attorney for Appellant(s)

December 3, 2008
Date of Signature

Respectfully submitted,



Forrest Gunnison
Attorney for Appellant(s)
Reg. No. 32,899
Tel.: (831) 655-0880

GUNNISON, McKAY & HODGSON, L.L.P.
Garden West Office Plaza
1900 Garden Road, Suite 220
Monterey, CA 93940
(831) 655-0880
Fax (831) 655-0888